pyTLEX: A Python Library for TimeLine EXtraction

Akul Singh, Jared Hummer, Mustafa Ocal, & Mark Finlayson
Florida International University
Knight Foundation School of Computing and Information Sciences
CASE Building, 11200 S.W. 8th Street, Miami, FL USA 33199
{asing118,jhumm001,mocal,markaf}@fiu.edu

Abstract

pyTLEX is an implementation of the TimeLine EXtraction algorithm (TLEX; Finlayson et al., 2021) that enables users to work with TimeML annotations and perform advanced temporal analysis, offering a comprehensive suite of features. TimeML is a standardized markup language for temporal information in text. pyTLEX allows users to parse TimeML annotations, construct TimeML graphs, and execute the TLEX algorithm to effect complete timeline extraction. In contrast to previous implementations (i.e., jTLEX for Java), pyTLEX sets itself apart with a range of advanced features. It introduces a React-based visualization system, enhancing the exploration of temporal data and the comprehension of temporal connections within textual information. Furthermore, pyTLEX incorporates an algorithm for increasing connectivity in temporal graphs, which identifies graph disconnectivity and recommends links based on temporal reasoning, thus enhancing the coherence of the graph representation. Additionally, pyTLEX includes a built-in validation algorithm, ensuring compliance with TimeML annotation guidelines, which is essential for maintaining data quality and reliability. pyTLEX equips researchers and developers with an extensive toolkit for temporal analysis, and its testing across various datasets validates its accuracy and reliability.

1 Introduction

Temporal information plays a crucial role in natural language processing and text analysis. TimeML, an SGML-based markup language, allows the annotation of temporal information in texts, including events, temporal expressions, links, and temporal signals (Pustejovsky et al., 2003a) [1]. TimeML annotations can be generated using automatic analyzers (Verhagen et al., 2005), manual annotation (Minard et al., 2016), or some combination of the two. TimeML annotations can be used to build temporal graphs, where nodes are events and temporal expressions, and edges are temporal relations. However, they provide only a partial ordering of events and times. Meanwhile, the global order (i.e., a timeline) is more useful for various NLP applications, including question-answering systems (Radev et al., 2002), text summarization (Mamidala and Sanampudi, 2021), and text visualization (Di Mascio et al., 2010).

To effect the extraction of timelines from TimeML annotations, we previously developed the TLEX algorithm, based on Constraint Satisfaction Problems (CSP), which provides an exact solution to the problem (in contrast to machine-learning-based approaches). TLEX converts TimeML annotations into an exact timeline, and in previous work, we introduced JTLEX, an open-source Java library, which implemented the TLEX algorithm (Ocal et al., 2023). jTLEX not only parsed TimeML annotations but also allowed users to manipulate TimeML graphs.

Like jTLEX, pyTLEX also takes a TimeML annotated file as input, then (1) parses the annotations into TimeML objects, (2) builds a TimeML graph, (3) partitions the TimeML graph into temporally connected graphs to separate real-life events and subordinated events, (4) transforms the temporally connected graphs into point algebra (PA) graphs, and (5) solves the PA graphs to extract a timeline. If a timeline cannot be extracted, meaning the graph is temporally inconsistent, (6) it detects the minimum inconsistent subgraph and returns it to the annotator to fix it. Finally, if the order of events and times are indeterminant (multiple possible ordering), (7) it calculates the temporal indeterminacy.

PyTLEX goes beyond jTLEX and introduces several new features. It includes a React-based application for graph and timeline visualization, making the exploration of temporal data more intuitive and insightful. The library incorporates an

[1] SGML is a markup language for defining the structure of documents in a machine-readable and human-readable format.

algorithm for automatically increasing connectivity, which detects graph disconnectivity and automatically suggests temporal links. Additionally, pyTLEX offers a rule-based system for validating compliance with the annotation guidelines.

We have tested pyTLEX on the TimeBank corpus (Pustejovsky et al., 2003b), which contains 183 TimeML annotated news articles. In less than 9 minutes on current consumer laptop (3.0 GHz Intel Core i7-1185G7 with 32GB of RAM), pyTLEX validated the annotations, extracted timelines, and visualized them. We release our demonstration system as well as a screencast video showing its operation[2].

2 Library Overview

2.1 User Input

pyTLEX offers comprehensive processing and manipulation capabilities for all the data present within a TimeML annotation. It accommodates various input sources, allowing the incorporation of TimeML annotations from a `.tml` file, a JSON-style TimeML encoding, or plain text. Users can also create TimeML annotations manually, adhering to the TimeML annotation guide (Sauri et al., 2006), or generate annotations automatically using advanced TimeML annotators like TARSQI (Verhagen et al., 2005), ClearTK (Bethard, 2013), CAEVO (Chambers et al., 2014), or CATENA (Mirza and Tonelli, 2016). It is important to note that automatic TimeML annotation tools, while efficient, may introduce limitations such as information loss, temporal inconsistencies, and incorrect annotations (Ocal et al., 2022a). The advantage of pyTLEX lies in its ability to detect and rectify such issues, as described in the subsequent sections.

2.2 TimeML Parser

pyTLEX offers a TimeML parser for transforming TimeML annotations into a collection of TimeML objects or the raw text. pyTLEX also validates annotation compliance with the standard.

2.3 Graph Constructor

In a TimeML graph, nodes correspond to events and times, and edges represent TimeML links, as illustrated in Figure 1. This graph encapsulates a wealth of information that can be programmatically queried, including sets of links and nodes, specific links by their ID, nodes by their ID, and lists of incoming or outgoing links, among other properties.

[2]https://cognac.cs.fiu.edu/pytlex/

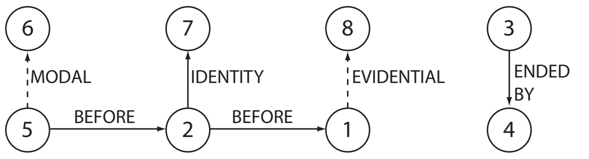

Figure 1: Visualization of the TimeML graph for `wsj_0555.tml` from the TimeBank corpus. SLINKs are given in dashed lines. pyTLEX partitions the TimeML graph into four temporally connected subgraphs.

PyTLEX also allows users to programmatically modify the TimeML graph. Users can introduce or remove links and nodes within the graph, allowing them to create custom graphs. The graph implementation can be exported as JSON, which can later be used for visualization. An example of a TimeML graph is given in Figure 1.

2.4 Partitioner

There are three types of TimeML links: <TLINK> and <ALINK> signify temporal order between events and times, while <SLINK> conveys modal, counterfactual, or conditional relationships between two events, as in the example "Tyler *forgot* to *bring* his wallet." In this instance, a counterfactual relationship exists between the events *forgot* and *bring*. The event *bring* never transpired in the "real world" described in the text. As detailed in the TLEX paper, pyTLEX partitions a TimeML graph into temporally connected subgraphs to identify such distinctions. The subgraph(s) containing "real world" events are called the *main* subgraph(s) and those connected to the main subgraphs via subordination links as *subordinated* subgraphs.

2.5 Transformer

As described in the TLEX algorithm, pyTLEX converts each temporally connected subgraph into a Point Algebra (PA) graph, where nodes are time points, and edges are primitive temporal constraints $<, =$. For example, if we have two events (A and B) with A being *BEFORE* B, this relationship is translated into a PA graph as $A^- < A^+ < B^- < B^+$, with '-' and '+' marking the start and end time points of a node. Figure 2 shows the PA graph for the TimeML graph in Figure 1. The PA graph is necessary for the temporal constraint satisfaction problem (TCSP) that is used to generate the

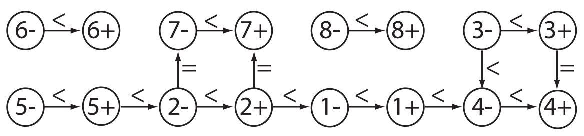

Figure 2: Visualization of the output of the transforming temporally connected subgraphs in Figure 1 into the PA graph after the connectivity increaser added the before link between 1 and 4.

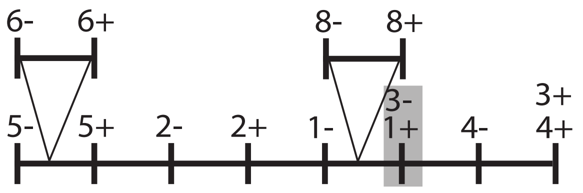

Figure 3: Visualization of the timeline of the TimeML graph in Figure 1. Grey regions indicate indeterminate sections.

timeline, as detailed in Section 2.6.

2.6 Solver

Once each temporally connected subgraph is transformed into a Point Algebra (PA) graph, pyTLEX uses the Z3 Python library for Constraint Satisfaction Problems (CSP) to assign integers to the time points within the graph. The timeline is then obtained by sorting these assigned integers. The Z3 Python library is a theorem prover and solver that is commonly used for solving complex mathematical and logical problems (De Moura and Bjørner, 2008). By default, pyTLEX generates the smallest solution where the first time point is assigned 1 and each subsequent time point the next lowest integer.

When applied to all the PA graphs, pyTLEX generates an exact trunk-and-branch timeline structure, where the trunk corresponds to the main timeline representing the main subgraph, and branches represent subordinated timelines associated with the subordinated subgraphs, as visualized in Figure 3. Therefore, the main timeline conveys the global order of "real world" events and times, while subordinated branches capture subordinated events. Users can extract various details from the timeline, such as its length, the first and last time points, the main timeline, subordinated branches, the count of subordinated branches, the number of time points, and the list of attachment time points where subordinated branches connect to the main timeline.

2.7 Inconsistency Detector

As described in the TLEX paper, the annotation must be consistent for the solver to extract a timeline. pyTLEX incorporates an inconsistency detection mechanism designed to identify inconsistent cycles in the TimeML graph. In such cases, pyTLEX identifies the specific links responsible for the inconsistency, thus enabling users to correct their annotations.

2.8 Indeterminacy Calculator

In many cases, natural language texts lack sufficient information to establish a unique ordering of events and times, resulting in multiple possible global orderings. As illustrated in Figure 2, there is no information regarding the relative order between $1+$ and $3-$. PyTLEX employs the TLEX algorithm to quantify temporal indeterminacy within a timeline. The algorithm explores and compares the shortest timeline with 100 alternative timelines (exhaustive computation of all possible timelines is computationally burdensome). If two adjacent points in the shortest timeline are not adjacent in all the other timelines, their order is indeterminate, and such sections can be marked as depicted in Figure 3.

2.9 Increasing Connectivity

During TimeML annotation, it's not uncommon for annotators to unintentionally overlook the annotation of temporal links. Such omissions can lead to disconnectivity within the TimeML graph, thereby disrupting the integrity of the timeline. pyTLEX integrates an algorithm detailed in (Ocal et al., 2022b) to address this problem. This algorithm leverages temporal reasoning to intelligently propose temporal links between two disconnected subgraphs. In essence, it undertakes a comparison of the temporal expressions within these subgraphs and, based on the evaluation of time values, automatically recommends the addition of temporal links. This not only streamlines the timeline generation process but also ensures the coherence and connectivity of temporal relationships within the annotated text. For example, in Figure 1, the TimeML graph is disconnected. Using the time values of 1 and 4, pyTLEX can suggest that 1 is BEFORE 4; inserting such a link results in a connected timeline, as shown in Figure 3.

2.10 Validation

The TimeML annotation guide (Sauri et al., 2006) establishes a set of rules governing the structure of TimeML annotations. pyTLEX incorporates these as a rule-based system that can scrutinize TimeML annotations and ensure compliance with those defined rules. Our validation system incorporates the algorithm presented in our prior work (Ocal et al., 2022b) to assess adherence to Rules 1 to 6. Furthermore, it adopts the algorithm outlined in (Derczynski and Gaizauskas, 2012) to verify compliance with Rule 7. Additionally, our system performs checks to identify instances of repeating links within the TimeML graph (Rule 8), reinforcing the integrity of the annotation.

2.11 Visualization

To visualize its JSON outputs, pyTLEX provides a React-based application that allows users to visually explore the TimeML graph, its partitions, and the resulting timelines. Additionally, the visualization application harnesses the output from the inconsistency detector to highlight problematic links within the graph. This visual aid empowers users to readily identify issues and undertake necessary corrections. Furthermore, pyTLEX incorporates visual cues to highlight indeterminate sections of the timeline, making it a valuable resource for narrative comprehension and understanding. For pyTLEX visualization, we have provided the demo video on the pyTLEX website [3].

3 Use Cases

A user guide and license information can be found on the pyTLEX website[4]. Here, we illustrate an approach for one of the TimeML annotations of the TimeBank corpus, called `wsj_0555.tml`. This file and the rest of the corpus can be obtained from the LDC website[5]. The following text, shown in the example below, is a snippet of the TimeML-annotated text of `wsj_0555.tml`. The TimeML graph corresponding to the snippet text is shown in Figure 1, where we can see that the nodes of the graph are either events or times, and the edges are TimeML relations. Event instance IDs and timeIDs are given in square brackets (DCT = DOCUMENT CREATION TIME).

[3]https://cognac.cs.fiu.edu/pytlex/
[4]https://cognac.cs.fiu.edu/pytlex/
[5]https://catalog.ldc.upenn.edu/LDC2006T08

[DCT:10/30/89$_{1[t12]}$]: Waxman Industries Inc. **said**$_{2[ei44]}$ holders of $6,542,000 face amount of its 6 1/4% convertible subordinated debentures, **due**$_{3[ei45]}$ **March** 15, 2007$_{4[t13]}$, have **elected**$_{5[ei46]}$ to **convert**$_{6[ei47]}$ the debt into about 683,000 common shares. The conversion price is $9.58 a share. The company **said**$_{7[ei48]}$ the holders **represent**$_{8[ei49]}$ 52% of the face amount of the debentures.

Users can read the file and create the TimeML graph as follows:

```
timeML_graph = Graph('wsj_0555.tml');
```

Users can retrieve any information about the graph, such as links (all or one by ID), nodes (all or one by ID), incoming links, outgoing links, JSON output, number of nodes, number of links, number of link types, etc. Listing 1 shows the output of pyTLEX when the user requests the information about the first link of wsj_0555.tml.

Moreover, users can actively manipulate their graph by adding or removing nodes and links or even constructing entirely custom graphs. The following code snippet demonstrates the process of creating a customized graph and adding two new nodes along with a link.

```
node1 = TimeX(1, "FUTURE_REF", True, "
    next wednesday")
node2 = TimeX(2, "FUTURE_REF", True, "
    next thursday")
link1 = Link(1, "TLINK", "BEFORE", node1
    , node2)
timeML_nodes = set()
timeML_nodes.add(node1)
timeML_nodes.add(node2)
timeML_links.add(link1)
timeML_graph = Graph(timeML_nodes,
    timeML_links)
```

After the TimeML graph is created, users can perform timeline extraction. Accessing the graph's partitions can be achieved as follows:

```
timeML_graph.main_partitions
timeML_graph.subordination_partitions
```

As can be seen from Figure 1, this TimeML graph has disconnectivity. PyTLEX can automatically propose a link based on the values of t12 (10/30/1989) and t10 (03/15/2007) through the use of the algorithm for increasing connectivity. Consequently, PyTLEX suggests the link "t12 –BEFORE-> t10" to the user. Users have the option to incorporate this suggested link, thereby achieving a fully connected graph and, by extension, a fully connected timeline:

```
Link: {ID = 1, LinkTag = TLINK, Syntax
      = "", Temporal Relation = BEFORE,
    Origin = null
    Related to time - Timex: {tID =
        t12, Type = DATE, Value =
        1989-10-30 , Mod = null,
        Temporal Function = true,
        Quantity = null, Frequency =
        null}
    Event Instance - Event Instance:
    {ID = eiid44, Tense = PAST, Aspect
         = NONE, Part of Speech = VERB,
         Polarity = POS, Modality = "
        null", Cardinality = "null",
        Signal = null
        EVENT: eid = e1, class =
            REPORTING, stem = say}
    }
```

Listing 1: pyTLEX parser output for printing the information about the first link of the graph.

```
Main Timeline: {
    eiid46- = 1
    eiid46+ = 2
    eiid48- = 3
    eiid44- = 3
    eiid44+ = 4
    eiid48+ = 4
       t12- = 5
       t12+ = 6
    eiid45- = 6
       t10- = 7
    eiid45+ = 8
       t10+ = 8
}
Attachment Points: {eiid46->eiid47,
    eiid48->eiid49}
Subordinated Timelines: {
[eiid47- = 1, eiid47+ = 2],
[eiid48- = 1, eiid48+ = 2]}
```

Listing 2: pyTLEX timeline output for the wsj_0555.tml file.

```
partitions = partition_graph(graph)
links = graph.links
Connectivity_Increaser.
    connect_partitions(partitions,len(
    links))
```

Now that we have the fully connected graph, we can extract the timeline. Users, can retrieve the *exact* trunk-and-branch timeline structure using:

```
timeML_graph.timeline
```

The output will be as shown in Listing 2. As can be seen, pyTLEX returns the *main* timeline, subordinated timelines, and the attachment points for each subordinated timeline.

After extracting the timeline, users can also retrieve the indeterminacy score, as well as the indeterminant time points. For our example, pyTLEX returns **0.125 indeterminacy score**, and **{t12+, eiid45-}** indeterminant time points after running:

```
IndeterminacyDetector.solve(g)
```

```
[Graph Type: Main Graph
Nodes Count = 2
Links count = 2
TLinkType: 2
ALinkType: 0
SLinkType: 0
Nodes:
eiid2048, t57
Links: (From -> To)
(t57 BEFORE eiid2048)
(eiid2048 BEFORE t57)
]
```

Listing 3: pyTLEX inconsistent subgraph output for the wsj_1011.tml file.

Users can validate annotations, for example, by checking the ALINK replacement rule (Rule 4) and the orphaned node rule (Rule 7):

```
filepath = r"../pytlex_data/
    TimeBankCorpus/wsj_0586.tml"
Sanity_Check.sco_ALINK_rule(filepath)
Sanity_Check.orphaned_node_rule(filepath
    )
```

Since the graph of wsj_0555.tml is consistent, pyTLEX's inconsistency detection method yields an empty set, indicating the absence of temporal inconsistencies. To elucidate the mechanics of the inconsistency detection algorithm, we can use wsj_1011.tml, which is a temporally inconsistent file from the TimeBank corpus. Following the execution of the graph construction method, users can run the generate_inconsistent_subgraphs(g) function to obtain information about the inconsistent cycle. For this specific file, pyTLEX generates an output as shown in Listing 3, presenting both the inconsistent subgraph and relevant subgraph details.

For pyTLEX visualization, we have provided the demo video on the pyTLEX website [6].

4 Related Work

As we discussed in Section 1, TimeML is a standardized temporal markup language. While numerous tools have been created for generating TimeML annotations, including (Verhagen et al., 2005; Saurí et al., 2005; Min et al., 2007; Chang and Manning, 2012; Chambers, 2013; Chambers et al., 2014; Bethard, 2013; Mirza and Tonelli, 2016), there are

[6]https://cognac.cs.fiu.edu/pytlex/

only a small number of tools designed to evaluate TimeML annotations.

Tango, a Java-based TimeML parser tool, can parse TimeML annotated documents and construct TimeML graphs (Verhagen et al., 2006). It offers users the ability to modify the graph and conducts temporal consistency checks using temporal closure. Tango was used in the evaluation of the TimeBank corpus, although it did not flag any inconsistencies within the TimeBank files. Notably, Tango employs <TIMEX> values to depict the graph in a timeline format, where each segment encompasses a <TIMEX> and the associated events. However, it does not furnish the global ordering of events.

Similarly, TBOX (Verhagen, 2007) can create a TimeML graph from TimeML annotations but eliminates temporal closure links. TBOX presents each event in this simplified graph as a box and positions these boxes based on their temporal relations, arranged according to their temporal order. This approach presents challenges with temporally indeterminate annotations.

Boguraev and Ando (2006) evaluated of the initial version of the TimeBank corpus, denoted as TimeBank 1.1. They analyzed the distribution of relations, event classes, Timex types, and TimeML components. Notably, their findings revealed that the annotation tool employed in the construction of TimeBank introduced a consistent shift of a single character. Additionally, they identified discrepancies in the types of Timex tags or instances of missing Timex tags associated with the same Timex signal in TimeBank 1.1. Boguraev et al. (2007) extended their analysis to TimeBank 1.2, enabling a comparative assessment. They selected a random document from both corpora and manually evaluated it to ascertain the number of errors in each version. Their results suggest that TimeBank 1.2 represented a notable improvement over 1.1.

TimeML-strict is a Java-based validation tool for TimeML (Derczynski et al., 2013). It implements a range of criteria, including the identification of any missing document creation time (DCT), the verification of non-linguistic content, and the confirmation that all links reference existing objects.

CAVaT is a Python utility for parsing TimeML annotations and providing quantitative insights, including distributions of TimeML objects (Derczynski and Gaizauskas, 2012). It offers functionality to detect self-loops, orphaned nodes, and disconnectivity within TimeML graphs. CAVaT excludes <SLINK>s from its analysis, which can result in subordinated events being erroneously identified as orphaned objects and subgraphs being marked as disconnected, despite their connectivity through <SLINK>s. CAVaT can identify temporal inconsistencies within the graph using closure. When temporal inconsistencies are present, CAVaT returns the most recently added constraint to the inconsistent cycle, which is important because identifying the complete inconsistent cycle based on a single edge can be particularly challenging for annotators, especially in the context of large graphs.

In addition to these TimeML tools, NLP researchers have used machine learning-based techniques for timeline extraction from TimeML annotations (Mani et al., 2006; Do et al., 2012; Kolomiyets et al., 2012; Leeuwenberg and Moens, 2020). These approaches come with specific constraints, and none of them address all temporal links, covering a maximum of 6 out of the 13 types. Additionally, they fail to distinguish between real-life events and subordinated events, and they may not effectively handle temporal indeterminacy within the annotations.

In contrast to prior work, pyTLEX provides an open-source implementation of TLEX, a technique for extracting exact timelines from TimeML annotations. Like its predecessor, jTLEX, pyTLEX incorporates a TimeML parser and a graph constructor. It distinguishes subordinated events from real-world events, extracts the global order of events and times in a trunk-and-branch timeline structure, automatically identifies and rectifies inconsistencies, and identifies and gauges indeterminacy. However, pyTLEX has a number of extended capabilities. It not only detects and resolves disconnectivities within both graphs and timelines but also integrates a validation system for TimeML annotations. Additionally, it offers a visualization system, enhancing comprehension for users.

5 Conclusion

We present pyTLEX, an open-source Python library that enables the programmatic extraction of exact timelines from TimeML-annotated texts via a standard Python API. PyTLEX provides several capabilities, including TimeML parsing, graph extraction, timeline generation, inconsistency identification, temporal indeterminacy assessment, disconnectivity detection and resolution, corpus validation, and advanced visualization capabilities. We release pyTLEX as an open-source library, freely available for non-commercial usage[7].

References

Steven Bethard. 2013. Cleartk-timeml: A minimalist approach to tempeval 2013. In *Second joint conference on lexical and computational semantics (* SEM), volume 2: proceedings of the seventh international workshop on semantic evaluation (SemEval 2013)*, pages 10–14.

Branimir Boguraev and Rie Kubota Ando. 2006. Analysis of timebank as a resource for timeml parsing. In *LREC*, pages 71–76.

Branimir Boguraev, James Pustejovsky, Rie Ando, and Marc Verhagen. 2007. Timebank evolution as a community resource for timeml parsing. *Language Resources and Evaluation*, 41:91–115.

Nathanael Chambers. 2013. Navytime: Event and time ordering from raw text. Technical report, NAVAL ACADEMY ANNAPOLIS MD.

Nathanael Chambers, Taylor Cassidy, Bill McDowell, and Steven Bethard. 2014. Dense event ordering with a multi-pass architecture. *Transactions of the Association for Computational Linguistics*, 2:273–284.

Angel X Chang and Christopher D Manning. 2012. Sutime: A library for recognizing and normalizing time expressions. In *Lrec*, volume 3735, page 3740.

Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer.

Leon Derczynski and Robert Gaizauskas. 2012. Analysing temporally annotated corpora with cavat. *arXiv preprint arXiv:1203.5051*.

Leon Derczynski, Hector Llorens, and Naushad UzZaman. 2013. Timeml-strict: clarifying temporal annotation. *arXiv preprint arXiv:1304.7289*.

T Di Mascio, R Gennari, I Lang, and P Vittorini. 2010. Visual tools for annotating temporal expressions with timeml: a critical overview. Technical Report KRDB10-5, KRDB Research Centre for Knowledge and Data, Free University of Bozen-Bolzano, Bolzano, Italy.

Quang Xuan Do, Wei Lu, and Dan Roth. 2012. Joint inference for event timeline construction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL'12)*, pages 677–687.

[7]`https://cognac.cs.fiu.edu/pytlex/`

Mark A Finlayson, Andres Cremisini, and Mustafa Ocal. 2021. Extracting and aligning timelines. In *Computational Analysis of Storylines: Making Sense of Events*, page 87. Cambridge University Press.

Oleksandr Kolomiyets, Steven Bethard, and Marie-Francine Moens. 2012. Extracting narrative timelines as temporal dependency structures. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL'12)*, pages 88–97.

Artuur Leeuwenberg and Marie-Francine Moens. 2020. Towards extracting absolute event timelines from english clinical reports. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28:2710–2719.

Kishore Kumar Mamidala and Suresh Kumar Sanampudi. 2021. A novel framework for multi-document temporal summarization (mdts). *Emerging Science Journal*, 5(2):184–190.

Inderjeet Mani, Marc Verhagen, Ben Wellner, Chong Min Lee, and James Pustejovsky. 2006. Machine learning of temporal relations. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics (ICCL-ACL'06)*, pages 753–760. Sydney, Australia.

Congmin Min, Munirathnam Srikanth, and Abraham Fowler. 2007. Lcc-te: a hybrid approach to temporal relation identification in news text. In *Proceedings of the fourth international workshop on semantic evaluations (SemEval-2007)*, pages 219–222.

Anne-Lyse Minard, Manuela Speranza, Ruben Urizar, Begona Altuna, Marieke Van Erp, Anneleen Schoen, and Chantal Van Son. 2016. Meantime, the newsreader multilingual event and time corpus. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 4417–4422.

Paramita Mirza and Sara Tonelli. 2016. Catena: Causal and temporal relation extraction from natural language texts. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 64–75.

Mustafa Ocal, Adrian Perez, Antonela Radas, and Mark Finlayson. 2022a. Holistic evaluation of automatic timeml annotators. In *Proceedings of the Language Resources and Evaluation Conference*, pages 1444–1453, Marseille, France. European Language Resources Association.

Mustafa Ocal, Antonela Radas, Jared Hummer, Karine Megerdoomian, and Mark Finlayson. 2022b. A comprehensive evaluation and correction of the timebank corpus. In *Proceedings of the Language Resources and Evaluation Conference*, pages 2919–2927, Marseille, France. European Language Resources Association.

Mustafa Ocal, Akul Singh, Jared Hummer, Antonela Radas, and Mark Finlayson. 2023. jTLEX: a Java library for TimeLine EXtraction. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 27–34, Dubrovnik, Croatia. Association for Computational Linguistics.

James Pustejovsky, José Castaño, Robert Ingria, Roser Saurí, Robert Gaizauskas, Andrea Setzer, and Graham Katz. 2003a. TimeML: robust specification of event and temporal expressions in text. In *Fifth International Workshop on Computational Semantics (IWCS-5)*, pages 1–11.

James Pustejovsky, Patrick Hanks, Roser Saurí, Andrew See, Rob Gaizauskas, Andrea Setzer, Dragomir Radev, Beth Sundheim, David Day, Lisa Ferro, and Marcia Lazo. 2003b. The TimeBank corpus. In *Proceedings of Corpus Linguistics Conference*, pages 647–656. Lancaster, UK.

Dragomir Radev, Beth Sundheim, Lisa Ferro, Roser Saurí, Andy See, and James Pustejovsky. 2002. Using timeml in question answering.

Roser Saurí, Robert Knippen, Marc Verhagen, and James Pustejovsky. 2005. Evita: a robust event recognizer for qa systems. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 700–707.

Roser Sauri, Jessica Littman, Robert Gaizauskas, Andrea Setzer, and James Pustejovsky. 2006. TimeML annotation guidelines, version 1.2.1. `https://catalog.ldc.upenn.edu/docs/LDC2006T08/timeml_annguide_1.2.1.pdf`.

Marc Verhagen. 2007. Drawing timeml relations with tbox. In *Annotating, extracting and reasoning about time and events*, pages 7–28. Springer.

Marc Verhagen, Robert Knippen, Inderjeet Mani, and James Pustejovsky. 2006. Annotation of temporal relations with tango. In *LREC*, pages 2249–2252.

Marc Verhagen, Inderjeet Mani, Roser Saurí, Jessica Littman, Robert Knippen, Seok Bae Jang, Anna Rumshisky, John Phillips, and James Pustejovsky. 2005. Automating temporal annotation with tarsqi. In *ACL*, pages 81–84.